LEARN C++
Skill up with our free tutorials

2.1 — Introduction to functions

ALEX[1] APRIL 9, 2024

In the last chapter, we defined a function as a collection of statements that execute sequentially. While that is certainly true, that definition doesn't provide much insight into why functions are useful. Let's update our definition: A **function** is a reusable sequence of statements designed to do a particular job.

You already know that every executable program must have a function named *main* (which is where the program starts execution when it is run). However, as programs start to get longer and longer, putting all the code inside the *main* function becomes increasingly hard to manage. Functions provide a way for us to split our programs into small, modular chunks that are easier to organize, test, and use. Most programs use many functions. The C++ standard library comes with plenty of already-written functions for you to use -- however, it's just as common to write your own. Functions that you write yourself are called **user-defined functions**.

Consider a case that might occur in real life: you're reading a book, when you remember you need to make a phone call. You put a bookmark in your book, make the phone call, and when you are done with the phone call, you return to the place you bookmarked and continue your book precisely where you left off.

C++ programs can work the same way. A program will be executing statements sequentially inside one function when it encounters a function call. A **function call** is an expression that tells the CPU to interrupt the current function and execute another function. The CPU "puts a bookmark" at the current point of execution, and then **calls** (executes) the function named in the function call. When the called function ends, the CPU returns back to the point it bookmarked, and resumes execution.

The function initiating the function call is the **caller**, and the function being called is the **callee** or **called** function.

An example of a user-defined function

First, let's start with the most basic syntax to define a user-defined function. For the next few lessons, all user-defined functions will take the following form:

```
returnType functionName() // This is the function header (tells the compiler about the existence of the function)
{
    // This is the function body (tells the compiler what the function does)
}
```

The first line is informally called the **function header**, and it tells the compiler about the existence of a function, the function's name, and some other information that we'll cover in future lessons (like the return type and parameter types).

- In this lesson, we'll use a *returnType* of *int* (for function *main()*) or *void* (otherwise). For now, don't worry about these, as we'll talk more about return types and return values in the next lesson (2.2 -- Function return values (value-returning functions)[2]).
- Just like variables have names, so do user-defined functions. The *functionName* is the name (identifier) of your user-defined function.
- The parentheses after the identifier tell the compiler that we're defining a function.

The curly braces and statements in-between are called the **function body**. This is where the statements that determine what your function does will go.

Here is a sample program that shows how a new function is defined and called:

```cpp
#include <iostream> // for std::cout

// Definition of user-defined function doPrint()
void doPrint() // doPrint() is the called function in this example
{
    std::cout << "In doPrint()\n";
}

// Definition of function main()
int main()
{
    std::cout << "Starting main()\n";
    doPrint(); // Interrupt main() by making a function call to doPrint().
main() is the caller.
    std::cout << "Ending main()\n"; // this statement is executed after
doPrint() ends

    return 0;
}
```

This program produces the following output:

```
Starting main()
In doPrint()
Ending main()
```

This program begins execution at the top of function *main*, and the first line to be executed prints `Starting main()`.

The second line in *main* is a function call to the function *doPrint*. We call function *doPrint* by appending a pair of parentheses to the function name like such: `doPrint()`. Note that if you forget the parentheses, your program may not compile (and if it does, the function will not be called).

Warning

Don't forget to include parentheses () after the function's name when making a function call.

Because a function call was made, execution of statements in *main* is suspended, and execution jumps to the top of called function *doPrint*. The first (and only) line in *doPrint* prints `In doPrint()`. When *doPrint* terminates, execution returns back to the caller (here: function *main*) and resumes from the point where it left off. Consequently, the next statement executed in *main* prints `Ending main()`.

Calling functions more than once

One useful thing about functions is that they can be called more than once. Here's a program that demonstrates this:

```cpp
#include <iostream> // for std::cout

void doPrint()
{
    std::cout << "In doPrint()\n";
}

// Definition of function main()
int main()
{
    std::cout << "Starting main()\n";
    doPrint(); // doPrint() called for the first time
    doPrint(); // doPrint() called for the second time
    std::cout << "Ending main()\n";

    return 0;
}
```

This program produces the following output:

```
Starting main()
In doPrint()
In doPrint()
Ending main()
```

Since *doPrint* gets called twice by *main*, *doPrint* executes twice, and *In doPrint()* gets printed twice (once for each call).

Functions can call functions that call other functions

You've already seen that function *main* can call another function (such as function *doPrint* in the example above). Any function can call any other function. In the following program, function *main* calls function *doA*, which calls function *doB*:

```cpp
#include <iostream> // for std::cout

void doB()
{
    std::cout << "In doB()\n";
}


void doA()
{
    std::cout << "Starting doA()\n";

    doB();

    std::cout << "Ending doA()\n";
}

// Definition of function main()
int main()
{
    std::cout << "Starting main()\n";

    doA();

    std::cout << "Ending main()\n";

    return 0;
}
```

This program produces the following output:

```
Starting main()
Starting doA()
In doB()
Ending doA()
Ending main()
```

Nested functions are not supported

Unlike some other programming languages, in C++, functions cannot be defined inside other functions. The following program is not legal:

```cpp
#include <iostream>

int main()
{
    void foo() // Illegal: this function is nested inside function main()
    {
        std::cout << "foo!\n";
    }

    foo(); // function call to foo()
    return 0;
}
```

The proper way to write the above program is:

```
#include <iostream>

void foo() // no longer inside of main()
{
    std::cout << "foo!\n";
}

int main()
{
    foo();
    return 0;
}
```

As an aside…

“foo” is a meaningless word that is often used as a placeholder name for a function or variable when the name is unimportant to the demonstration of some concept. Such words are called metasyntactic variables (https://en.wikipedia.org/wiki/Metasyntactic_variable)[3] (though in common language they're often called “placeholder names” since nobody can remember the term “metasyntactic variable”). Other common metasyntactic variables in C++ include “bar”, “baz”, and 3-letter words that end in “oo”, such as “goo”, “moo”, and “boo”).

For those interested in etymology (how words evolve), RFC 3092 (https://datatracker.ietf.org/doc/html/rfc3092)[4] is an interesting read.

Quiz time

Question #1

In a function definition, what are the curly braces and statements in-between called?

Hide Solution (javascript:void(0))[5]

> The function body

Question #2

What does the following program print? Do not compile this program, just trace the code yourself.

```cpp
#include <iostream> // for std::cout

void doB()
{
    std::cout << "In doB()\n";
}

void doA()
{
    std::cout << "In doA()\n";

    doB();
}

// Definition of function main()
int main()
{
    std::cout << "Starting main()\n";

    doA();
    doB();

    std::cout << "Ending main()\n";

    return 0;
}
```

Hide Solution (javascript:void(0))[5]

```
Starting main()
In doA()
In doB()
In doB()
Ending main()
```

Next lesson
2.2 Function return values (value-returning functions)

2

Back to table of contents

6

Previous lesson
1.x Chapter 1 summary and quiz

7

8

B U URL INLINE CODE C++ CODE BLOCK HELP!

Leave a comment...

Name*

Email*

Notify me about replies:

POST COMMENT

Find a mistake? Leave a comment above!

Avatars from https://gravatar.com/[11] are connected to your provided email address.

775 COMMENTS Newest

tamim-san
April 8, 2024 10:03 pm

how did Question3
print
doA first??

0 Reply

rookie
Reply to tamim-san [12] April 9, 2024 6:11 am

Because the main function first calls the doA function

0 Reply

tamim-san
Reply to rookie [13] April 9, 2024 8:52 am

I see thanks.

0 Reply

Amir

April 8, 2024 6:47 am

Think would be ok to tell the purpose of 'void', since we are already familiar with 'return'.
Also wouldn't it be more appropriate to number the lesson 2.0 not 2.1.
Thnx a lot!

0 Reply

Alex Author

Reply to Amir 14 April 9, 2024 8:12 am

We cover `void` in a few lessons.

Does it really matter whether the tutorials are 1-based or 0-based?

0 Reply

Jawad Ahmad

March 3, 2024 10:58 pm

Very very impressive and informative but I would suggest to reconsider the statement "Functions calling functions calling functions", because I understood it after finishing the whole topic. It ia a little bit confusing. It would be better to use the statement, " Callee functions being caller functions" or "Callee Functions calling other Functions" or "user-defined functions as caller functions".
Thanks a Lot!

0 Reply

Alex Author

Reply to Jawad Ahmad 15 March 5, 2024 1:36 pm

Section name updated to "Functions can call functions that call other functions". Using "callee" and "caller" here is difficult since the middle function is both a callee and a caller.

5 Reply

Jawad Ahmad

Reply to Alex 16 March 5, 2024 5:49 pm

Thanks a lot

0 Reply

nava

February 17, 2024 11:43 am

why isn't it ?

```
In doB()
In doA()
In doB()
starting main
In doA()
In doB()
In doB()
ending main
```

Is it that void is used as neutral, like it declares/ put code but isn't used until called in main, I didn't saw explication for void or i don't remember (maybe)?

0 Reply

tamim-san

Reply to nava 17 April 8, 2024 10:11 pm

No that's what letme compile it.

`Starting main()`
`In doA()`
`In doB()`
`In doB()`
`Ending main()`

nope idk ig they'll explain in the chapter.

0 Reply

Alex Author

Reply to nava 17 February 17, 2024 6:36 pm

Execution doesn't start at the top of the file -- it starts at the top of the main() function.

5 Reply

nava

Reply to Alex 18 February 28, 2024 1:26 am

okk thanks a lot, I didn't understand it

0 Reply

cake

Reply to nava 19 February 28, 2024 11:15 am

What Alex said is, whatever code you write off, the code execution (by the compiler) will always start with the first line of the main function, then next line, and so on. When we write some statement like "dob()" inside main(), it means that it will jump to that fuction ("dob()") and execute it line-wise until the end of function (eof) and the again jump back to main() and execute next line.

For 'void', first of all it's a datatype just like 'int' but it's importance is that unlike 'int' functions (like commonly main() function) which gives some return value (mostly "return 0", which tells successful execution of program), with 'void' function (like "void main()") you don't need to return anything.

Note: There can be integer variables, but not of the type 'void'. 'void' datatype is designed like it's an incomplete type that cannot be completed.

Let's not complicate it any further :)
Just remember this much

Last edited 1 month ago by cake

1 Reply

nava

Reply to cake 20 February 28, 2024 10:19 pm

alright, thanks for the enlightenment. while understanding the void function and the call of functions that jumps, I got something else to ask now:

in the previous chapters, it said a code has to have a main function, but does it always have to be at the end of the code ? like can you define functions after the main function is developed ?

0 Reply

Alex Author

Reply to nava 21 March 4, 2024 11:28 am

main() is often at the bottom because it's most convenient to place it there -- that way any other functions it calls will already have been defined.

However, you can define main() anywhere in your file, and use forward declarations to tell the compiler about the existence of functions that are defined later in the file. This is covered later in this chapter.

2 Reply

according_ad

February 16, 2024 12:47 am

why is this not working? it says C3861 'doB': identifier not found

```cpp
#include <iostream>


void doA()
{
    std::cout << "starting 2 \n";
    doB();
}

void doB()
{
    std::cout << "starting 3 \n";
}

int main()
{
    std::cout << "starting 1 \n";

    doA();

    std::cout << "ending at 4 \n";

    return 0;
}
```

0 Reply

Kamil G.

Reply to according_ad [22] March 18, 2024 12:01 pm

Do `doB()` above `doA()` or forward declaration of `doB()` above `doA()`.

Like that:

```
#include <iostream>

void doB(); //forvarding declaration of doB()
void doA()
{
    std::cout << "starting 2 \n";
    doB();
}

void doB()
{
    std::cout << "starting 3 \n";
}

int main()
{
    std::cout << "starting 1 \n";

    doA();

    std::cout << "ending at 4 \n";

    return 0;
}
```

1 Reply

Martin
Reply to according_ad[22] February 16, 2024 11:36 am

Solution: put the lines for doB() **above** doA().

Reason: in doA() you use doB(), but it is not defined yet.

3 Reply

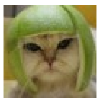
Matthew
January 19, 2024 8:59 am

```cpp
#include <iostream> // for std::cout

void doPrint();

// Definition of function main()
int main()
{
    std::cout << "Starting main()\n";
    doPrint(); // Interrupt main() by making a function call to doPrint(). main() is the caller.
    doPrint();
    std::cout << "Ending main()\n"; // this statement is executed after doPrint() ends

    return 0;
}

// Definition of user-defined function doPrint()
void doPrint() // doPrint() is the called function in this example
{
    std::cout << "In doPrint()\n";
}
```

Why not declare just the function name above main(), and then place the actual code for the user function below main?

Last edited 2 months ago by Matthew

0 Reply



Alex Author
Reply to Matthew 23 January 19, 2024 9:52 am

Because we haven't covered forward declarations yet.

11 Reply

Swaminathan
January 17, 2024 5:49 am

Alex, you are to people like me what St.Paul was for christianity ..
My love for C++ and in general other languages that I know has grown a lot..Many Thanks!

1 Reply

Adidaphat
December 31, 2023 10:38 pm

Hello, when can i study data structures and algorithms? I have studied some basic algorithms, but they are very difficult,.. I'm tired of it

1 Reply

AbeerOrTwo
December 14, 2023 8:18 pm

twas a good lesson

0 Reply

0ld Camel
December 6, 2023 1:30 am

When I started on 04 Dec 2023, the website formated better i.e. colored boxes (green for should, yellow and red etc...) All these formating seems to disappeared now.

0 Reply

Sevgi
November 20, 2023 2:39 pm

topics are EZ so far

0 Reply

> **joe**
> Reply to Sevgi 24 January 12, 2024 7:09 am
>
> ez money
>
> 0 Reply

ClickBrick
October 16, 2023 3:24 pm

This is a little practice code I did after the lesson.

#include <iostream>

void billy()
{
std::cout << "Help, I am a function inside of a function!\n";
}

void clemon()
{
std:: cout << "Something is inside of me main()\n";

billy();
}

int main()
{
std::cout << "Hi Billy\n";
clemon();

return 0;
}

2 Reply

new User
October 13, 2023 9:10 am

```
#include <iostream>

int main();

void doA()
{
    std::cout << "inside doA() \n";
    main();
}

int main()
{
    std::cout << "Starting main()\n";

    doA();

    std::cout << "Ending main()\n";
    return 0;
}
```

Anyone tried this code?;

2 Reply

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Reply to new User 25 December 24, 2023 10:20 am

```
#include <iostream>

int main()
{
    std::cout << "a";
    main();
    return 0;
}
```

2 Reply

Tom

Reply to aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa [26] January 7, 2024 6:19 am

```cpp
#include <iostream>

int count { 5 };

int main() {
    if (count > 0) {
        std::cout << count << '\n';
        count = count - 1;
        main();
    } else {
        return 0;
    }
}
```

finite loop

1 Reply

osnam

Reply to new User [25] October 28, 2023 1:27 am

surprised.
I ran this and it returns the printed statements in a repeated loop. I was expecting nothing to be returned since the first declaration of "main" has an empty function body.

0 Reply

Pratyush

Reply to new User [25] October 16, 2023 11:39 pm

yeah a endless loop.

0 Reply

ClickBrick

Reply to new User [25] October 16, 2023 3:27 pm

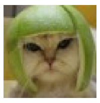
That is pretty clever.

0 Reply

azizkurtariciniz
September 18, 2023 8:45 am

Go for the eyes, Boo, go for the eyes!

6 Reply



Alex Author
Reply to azizkurtariciniz [27] September 21, 2023 8:44 am

RrraaaAAGHGHH!

4 Reply

Gamborg
September 13, 2023 5:11 am

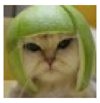
So I have been going through the previous chapters by now. And I have been playing around with the topics from these. After reading this chapter I thought it would be fun to put everything I did previously into their own functions. So as to keep main() "simple". This is the monstrosity I have atm...

```cpp
#include <iostream>

void checkEquality() {

    int equalityOne{};
    int equalityTwo{};

    std::cout << "          This program will check if two intergers are equal to each other          " << '\n';
    std::cout << "Enter first integer: ";
    std::cin >> equalityOne;
    std::cout << "Enter second integer: ";
    std::cin >> equalityTwo;
    std::cout << '\n' << "The two numbers which were entered were " << equalityOne << " and " << equalityTwo << '\n';

    if (equalityOne == equalityTwo) {
        std::cout << "The two numbers are equal in value";
    }
    else {
        std::cout << "The two numbers are not equal in value";
    }
}

void doubleInteger() {

    int num{};
    int num2{};

    std::cout << "Enter an integer: ";

    std::cin >> num;

    std::cout << "You entered: " << num << '\n';

    std::cout << "The double of what you entered is: " << num * 2 << '\n' <<
'\n';
}

void doubleAndTripleInteger() {

    int x{};

    std::cout << "Enter an integer: ";

    std::cin >> x;

    std::cout << "Double " << x << " is: " << x * 2 << '\n';

    std::cout << "Triple " << x << " is: " << x * 3 << '\n' << '\n';

}

void anotherRandomIntegerHandling() {

    int multiplyNum{};
    int multiplyNum2{};

    std::cout << '\n' << '\n' << "Enter an integer: ";
    std::cin >> multiplyNum;
    std::cout << "Enter a second integer: ";
    std::cin >> multiplyNum2;
    std::cout << multiplyNum << " + " << multiplyNum2 << " is " << multiplyNum + multiplyNum2 << "." << '\n';
    std::cout << multiplyNum << " - " << multiplyNum2 << " is " << multiplyNum - multiplyNum2 << "." << '\n';
    multiplyNum - multiplyNum2 << ' ' << '\n';
}

int main()
{
    doubleInteger();

    doubleAndTripleInteger();

    checkEquality();

    anotherRandomIntegerHandling();

    return 0;
}
```

Just some playing around with numbers from previous chapters. I am sure there are things I could do better.

I did notice however, if I tried to enter a decimal number into any of the list initialized values, it would reduce to the nearest integer value. I thought we used list initialization of values to prevent this from happening at all? That if we tried to input non integer values, it would throw errors?

0 Reply



Alex Author

Reply to Gamborg 28 September 15, 2023 12:03 pm

List initialization only prevents narrowing conversions at the point of initialization. It does not have any effect after that, such as when inputting a value using `operator>>`.

3 Reply

nywerya

July 13, 2023 8:16 pm

非常好□程

2 Reply

Nobody

June 22, 2023 2:01 pm

I feel like I have an okay understanding of functions!

```
#include <iostream>
//Here is a simple program that will explain the process of the functions
doA(), doB() and main()
void doB()
{
    std::cout << "doB() completes it's task, then goes back to doA()\n";
}
void doA()
{
    std::cout << "doA() stops to call to doB()\n";
    doB();
    std::cout << "After doB() is finished doA() completes it's task and back
to main()\n";
}
int main()
{
    std::cout << "main() function stops to call doA()\n";
    doA();
    std::cout << "main() function completes it's task and finishes.\n";
    return 0;
}
```

3 Reply

nywerya
Reply to Nobody 29 July 13, 2023 8:16 pm

sure

0 Reply

Parka
Reply to Nobody 29 July 3, 2023 3:07 pm

Yessir!

0 Reply

fitz
June 14, 2023 12:04 pm

this course is so much more defining than other courses ive taken so far. i just wanted to say thank you!

3 Reply

Platzhalten
April 14, 2023 7:52 am

When i run this:

```
int main()
{
std::cout << "Test start";
test();
std::cout << "Test end";
}

int test()
{
std::cout << "Hello";
}
```

i get the error: "test": Identifier not found
with the Error code C3861

Last edited 1 year ago by Platzhalten

0 Reply

Stefanos

Reply to Platzhalten 30 February 6, 2024 10:34 am

Sorry but i couldn't hold back my laugher when i saw your code .
It seems you didnt even read the article when you posted this comment back in the day .
XD

0 Reply

Platzhalten

Reply to Stefanos 31 February 7, 2024 8:34 pm

i think i read only like every 2. word so you right i guess

0 Reply

Hoyt

Reply to Platzhalten 30 April 14, 2023 11:48 am

The test() method must appear above main otherwise the compiler doesn't recognize it. (This is different from other languages like Java where methods can appear below their caller).

Last edited 1 year ago by Hoyt

0 Reply

Platzhalten

Reply to Hoyt[32] April 16, 2023 2:06 pm

bruh

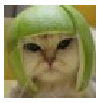
1 Reply

int test = wrong

Reply to Platzhalten[33] April 29, 2023 2:15 pm

int is used for integers

0 Reply



Alex Author

Reply to int test = wrong[34] May 2, 2023 10:42 pm

int is the return type of function test() in this case.

1 Reply

Michael Murphy

December 11, 2022 4:25 pm

The first time I ran into the "foo" "bar" combination, I just figured the programmer had forgotten the correct spelling for FUBAR.

Last edited 1 year ago by Michael Murphy

6 Reply

Links

1. https://www.learncpp.com/author/Alex/
2. https://www.learncpp.com/cpp-tutorial/function-return-values-value-returning-functions/
3. https://en.wikipedia.org/wiki/Metasyntactic_variable
4. https://datatracker.ietf.org/doc/html/rfc3092
5. javascript:void(0)
6. https://www.learncpp.com/
7. https://www.learncpp.com/cpp-tutorial/chapter-1-summary-and-quiz/

8. https://www.learncpp.com/introduction-to-functions/
9. https://www.learncpp.com/cpp-tutorial/introduction-to-objects-and-variables/
10. https://www.learncpp.com/wordpress/recent-news-box-on-front-page-for-tiga-102-theme/
11. https://gravatar.com/
12. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-595569
13. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-595585
14. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-595529
15. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-594243
16. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-594321
17. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-593747
18. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-593771
19. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-594133
20. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-594140
21. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-594153
22. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-593684
23. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-592573
24. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-590059
25. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-588743
26. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-591322
27. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-587369
28. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-587122
29. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-582300
30. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-579300
31. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-593315
32. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-579309
33. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-579389
34. https://www.learncpp.com/cpp-tutorial/introduction-to-functions/#comment-579912